Ramesh

Q — Grammar

Version 0.1

May 8, 2015

Contents

Q — Grammar

1 Introduction

Q is a vector language, designed for efficient implementation of counting, sorting and data transformations. It uses a single data structure — a table.

1.1 Motivation

I will motivate the need for Q by quoting from two of my Gods — Codd and Iverson. I could be accused of quoting scripture for my purpose (see below) and it is true that I am being selective in my extracts. However, that does not detract from their essential verity.

> The devil can cite Scripture for his purpose.
> An evil soul producing holy witness
> Is like a villain with a smiling cheek,
> A goodly apple rotten at the heart:

1.1.1 Extracts from Codd

The most important motivation for the research work that resulted in the relational model was the objective of providing a sharp and clear boundary between the logical and physical aspects of database management. We call this the *data independence objective*.

A second objective was to make the model structurally simple, so that all kinds of users and programmers could have a common understanding of the data, and could therefore communicate with one another about the database. We call this the *communicability objective*.

A third objective as to introduce high level language concepts *but not specific syntax) to enable users to express operations upon large chunks of information at a time. This entailed providing a foundation for set-oriented processing (i.e., the ability to express in a single statement the processing of multiple sets of records at a time). We call this the *set-processing objective*.

To satisfy these three objectives, it was necessary to discard all those data structuring concepts (e.g., repeating groups, linked structures) that were not familiar to end users and to take a fresh look at the addressing of data.

We have deviated from Codd's preference for the relational model. Instead, we choose to drop down one level to the table. As Codd writes:

Tables are at a lower level of abstraction than relations, since they give the impression that positional (array-type) addressing is applicable (which is not true of n-ary relations), and they fail to show that the information content of a table is independent of row order. Nevertheless, even with these minor flaws, tables are the most important conceptual representation of relations, because they are universally understood.

Lastly, in designing Q, we wanted it to be a data model as Codd defines one

A data model is a combination of at least three components:

1. A collection of data structure types (the building blocks);
2. A collection of operators or rules of inference, which can be applied to any valid instances of the data types listed in (1), to retrieve, derive, or modify data from any parts of those structures in any combinations desired;
3. A collection of general integrity rules, which implicitly or explicitly define the set of consistent database states or changes of states or both

1.1.2 Extracts from Iverson

The importance of language has been stated over the centuries. Iverson quotes the following from Whitehead:

By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and in effect increases the mental power of the race

In the same vein, he quotes Babbage:

The quantity of meaning compressed into small space by algebraic signs, is another circumstance that facilitates the reasonings we are accustomed to carry on by their aid

I would hesitate to claim that Q meets any of the following criteria that Iverson lays down for good notation. But it is definitely the guiding principle and aspirational goal for Q.

1. Ease of expressing constructs arising in problems. If it is to be effective as a tool of thought, a notation must allow convenient expression not only of notions arising directly from a problem, but also of those arising in subsequent analysis, generalization and specialization.

2. Suggestivity. A notation will be said to be suggestive if the forms of the expressions arising in one set of problems suggests related expressions which find application in other problems.

3. Ability to subordinate detail. Brevity is achieved by subordinating detail, and we will consider three important ways of doing this

   - the use of arrays
   - the assignment of names to functions and variables
   - the use of operators

4. Economy. Economy requires that a large number of ideas be expressible in terms of a relatively small vocabulary.

5. Amenability to formal proofs

2 Notations and Conventions

2.1 Auxiliary Fields

An auxiliary field is a field that belongs to a primary field. It is used to create some property for the priomary field

1. NullFld. This is a boolean field which tells us whether the corresponding entry in the primary field is null or not

2. LenFld. This is an integer field which tells us the length of the corresponding entry of the primary field, whose FldType is SV

3. OffFld. The data for an SV field is stored as a continuous sequence of bytes. The value of this field tells us how far the text for this cell is from the start.

Definition 1 *A boolean field is an I1 field with* $\mathrm{MinVal} \geq 0$ *and* $\mathrm{MaxVal} \leq 1$

2.2 Environment Variables

Q_META_DATA Name of file where meta data is stored

Q_DATA_DIR Name of directory where data is stored. Needs to be unique for each table space.

Q_RUN_TIME_CHECKS If it is set, then we execute run time.

2.3 Notations

- The upside down T symbol $\perp$ means null
- Properties are of the form `PROP=foobar` where `foobar` is the desired property. Foobar must be alphabetic.
- Operators are of the form `OP=foobar` where `foobar` is the desired operator. Foobar must be alphabetic.
- Args are of the form `ARGS={valid json}`. Limitations on the characters permitted in the JSON? **TO BE COMPLETED**
- Wherever it is possible to have an I1 field, f, on the RHS of a Q statement, one can have the nn field, $NN(f)$. Are there any exceptions to this rule? **TO BE COMPLETED**
- When a string is used for user input (like a regular expression match), it is uuencoded on the way in so that there is no conflict with other non-alpha numeric characters which have special meaning to the parser
- Lower bounds are inclusive and upper bounds are exclusive.
- If an operation is expected to create a table T and a table by that name exists, then the original table is first deleted. Examples of operations that create tables are **TO BE COMPLETED**
- If an operation is expected to create a field with name f and a field with such a name exists, then the old field is deleted **after** the operation is performed and replaced by the newly created field.

- A table has a non-null, unique name.
- A field has a non-null name that is unique within a table.
- If a field, `foo` has null values, then property **HasNullFld** is true and a boolean field tt .nn.foo exists.
- Most operations cannot be performed on the auxiliary field of a primary field. When this is possible, we will say so explicitly.

2.4 Field Types

See Table 1. Details below

LABELS Assume $T.f$ is a string field. However, assume that the number of unique strings is relatively small and our use of this string is just as a label. In such cases, we will create a "lookup table", T_L, that has a field **txt**. We will set $T.f$ to be an integer field, serving as a foreign key to T_L. Hence, $0 \leq T.f[i] < |T_L|$, We set property `LkpTbl` tp T_L and proeprty `LkpFld` to `txt`. For purposes of comparison, we use integer comparisons, which is much faster. However, when we have to print out the value, we need an indirection step to access the string corresponding to the integer.

SC We store one byte (for null character) more than what user specifies as length. Can have a null field. Verify this **TO BE COMPLETED**

SV **TO BE COMPLETED**

2.5 Some Useful Enumerations

Invariant 1 $\text{FldType} = SC \Leftrightarrow \text{Width} \neq \perp$

Invariant 2 $\text{FldType} = SV \Leftrightarrow \text{HasOffFld} = true$

Invariant 3 $\text{FldType} = SV \Leftrightarrow \text{HasLenFld} = true$

Invariant 4 $\text{LkpTbl} \neq \perp \Leftrightarrow \text{LkpFld} \neq \perp$

Invariant 5 $\text{LkpTbl} \neq \perp \Rightarrow \text{FldType} \in \{I1, I2, I4, I8\}$

Value	Description	Status
I1	1 byte signed integer	✓
I2	2 byte signed integer	✓
I4	4 byte signed integer	✓
I8	8 byte signed integer	✓
F4	4 byte floating point	✓
F8	8 byte floating point	✓
SC	constant length string	✓
F8	variable length string	✓
B	bit field	WIP

Table 1: Types supported by Q

Value
Unknown
Ascending
Descending
Unsorted

Table 2: Return Values for SortType

Abbreviation	Explanation
NumRows	number of rows
Exists	whether it exists or not
RefCount	number of tables using it

Table 3: Properties of Tables

Abbreviation	Explanation
HasNullFld	does it have a null field
HasLenFld	does it have a length field
HasoffFld	does it have an offset field
FldType	one of values in Section 2.4
SortType	one of values in Table 2
LkpTbl	lookup table
Lkpfld	lookup field
Width	width for FldType = SC

Table 4: Properties of Fields

2.6 PTO — Partial Table Operations

There are three ways in whicha partial table can be specified. They are all writte as $T_1|(\ldots)$ where the $\ldots$ can be one of

1. a boolean field, f_c. In this case, we consider only rows of T_1 where $f_c = 1$
2. n_1, n_2 where $0 \leq n_1 < n_2 < n_R$, where n_R is the number of rows in the table. In this case, we consider only those rows whose indexes are $\geq n_1 \wedge < n_2$ boolean (I1 to be precise) field has value 1
3. T_2, f_{lb}, f_{ub}. In this case, f_{lb}, f_{ub} are I8 fields in T_2 such that $f_{lb} < f_{ub}$. The i^{th} row of T_1 is considered only if $\exists j : T_2[j].f_{lb} \leq i < T_2[j].f_{ub}$

2.7 Pragmas

There are times when we wish to

1. modify the normal behavior of an operation
2. provide hints to the execution engine as to which algorithm to use

These are provided as **pragmas** in the command, which take the form `PRAGMA=JSON STRING`

3 Commands

See Table 5

Command	**Summary**	**Details**
$+$ Scope S	Start Scope S	
$-$ Scope	Stop Current Scope	
? Scope	Print Scope Hierarchy	
$+$ Compound	Start Compound Expression	
$-$ Compound	Stop Compound Expression	
OP=none	no op	
OP=dump FileName	save meta data in file	Section 4.15
$\#T$ $?T$ $?T*$ $?T$ **PROP=x**	number of rows in table does table exist all meta data for table describe property x of table	 Table 3
$?T.f$ $?T.f*$ $?T.f$ **PROP=x**	is field in table all meta data for field describe property x of field $T.f$	 Table 4
$+T.f$ **PROP=x** v $+T_1.f_1$ **PROP=nnfld** $T_2.f_2$	set property x of field $T.f$ to v THINK	Table 4 THINK
$-T.f$ **PROP=x**	unset property x of field $T > f$ of table T	
$T :=$ **OP=New** n $T :=$ **OP=LoadCSV** $\mathcal{A}$ $T :=$ **OP=LoadBin** $\mathcal{A}$ $T :=$ **OP=LoadHDFS** $\mathcal{A}$	add empty table with n rows load from CSV load from binary load from hdfs	 Section 3.3 Section 3.3 Section 3.3
$T.f :=$ **OP=LoadBin** $\mathcal{A}$ $T.f :=$ **OP=LoadCSV** $\mathcal{A}$	load single field load single field	Section 3.2 Section 3.2
$-T$ $-\{T_1, T_2, \ldots\}$ $-T.f$ $-T.\{f_1, f_2, \ldots\}$ $-T.f$ **PROP=NNFld**	delete table delete tables delete field delete fields THINK	 THINK
$T.f :=$ **OP=** $\oplus$ $\mathcal{A}$	scalar to field	Section 3.4
OP= $\oplus$ $T.f$ **OP=** $\oplus$ $T\|(\ldots).f$ $T_1 \leftarrow T_2$	field to scalar field to scalar rename table	Section 3.1 Section 3.1 Section **??**
Q Commands		*continued on next page*

Command	**Summary**	**Details**
$T_1 := T_2$ $T_1 := T_2\|(..)$	copy table copy partial table	Section 4.8 Section 4.8
$T_1 \stackrel{+}{=} T_2$	append table T_2 to T_1	Section 4.4
$T_1.f_1 \leftarrow T_2.f_2$	replace f_1 in T_1 with f_2 in T_2 and delete f_2 in T_2	
$T_1.f_1 := T_2.f_2$ $T_1.f_1 := T_2\|(\ldots).f_2$	copy field copy partial field	Section 4.9 Section 4.9
$T_1(f_1 := f_2)$	duplicate field	
$T_1(f_1 <= f_2)$	rename field	After, no $T_1.f_2$
$T_1.f_1 := T_2.f_2$	duplicate field	After, no $T_2.f_2$
$T_1.f_1 <= T_2.f_2$	move field	
$T_1.NN(f_1) := T_2.f_2$	make null field	THINK
$T_1.f_1 := T_2.NN(f_2)$	break null field	THINK
$T_1(f_1 :=$ **OP=** $\oplus\ f_2)\ \mathcal{A}$	Create f_1 from f_2	Section 3.5
$T_1(f_1 :=$ **OP=** $\oplus\ \ \mathcal{A} f_2)$	Create f_1 from scalar and f_2	Section 3.6
$T_1.f_1 :=$ **OP=** $\oplus\ T_2.f_2\ \mathcal{A}$	Create $T_1.f_1$ from $T_2.f_2$	
$T_1(f_1 := f_2$ **OP=** $\oplus\ f_3)\ \mathcal{A}$	Create f_1 form f_2, f_3	Section 3.8
$T_1(w :=?:\ \ x\ \ y\ \ z)$ $T_1(w :=?:\ \ lb\ \ ub\ \ y\ \ z)$ $T_1(w :=?:\ \ lb\ \ ub\ \ val)$	ternary operator ternary operator set value	Section 3.7 Section 3.7 Section 3.7
$T_1.\{f_I, f_1\} :=$ **OP=Permute** $T_2.f_2$	permute	Section 4
$T_1.f_0 :=$ **OP=Pack** $T_1.\{f_1, f_2, f_3, \ldots\}\ \mathcal{A}$ $T_1.\{f_1, f_2, \ldots\} :=$ **OP=UnPack** $T_1.f_0\ \mathcal{A}$ $T_1.\{f_1, f_2\} :=$ **OP=UnConcat** $T_2.f_0$	pack many fields into one Section 4.1 unpack one field into many unpack 2 fields into one	 Section **??** Section 4.10
$T_1.f_1 :=$ **OP=** $\oplus\ T_2.f_2\ \ T_3.f_3$	$\cup, \cap, -$	Section 4.7
OP=Sort $T.f\ \mathcal{A}$ **OP=Saturate** $T.f\ \mathcal{A}$ **OP=Permute** $T.f\ \mathcal{A}$	in place sort in place saturate in place random permute	Section 4.2
OP=Sort $T.\{f_1, f_2\}\ \mathcal{A}$	in place joint sort	Section 4.3
$T_1.\{f_L, f_U, f_n, f_\mu\} :=$ **OP=Quantiles** $T_2.f_2\ \mathcal{A}$	Calculate quantiles Calculate approximate quantiles	Section 4.5
$T_1.\{f_L, f_U, f_C\} :=$ **OP=NumInRange** $T_2.f_2$	count number in range	Section 4.6
$T_1.\{f_v, f_n\} :=$ **OP=CountValues** $T_2.f_2\ \mathcal{A}$	count number of occurrences	Section 4.13
Q Commands		*continued on next page*

Command	**Summary**	**Details**
$T_1.\{f_v, f_n\}$:= **OP=CountValues** $T_2\|(\ldots).f_2\ \mathcal{A}$	count number of occurrences	Section 4.13
$T_D.\{l_D, f_D\}$:= **OP=Join** $T_S.\{f_S, l_S\}\ \mathcal{A}$	join	Section 4.14
$T_D.f_D$:= **OP=Top** $T_S.f_S\ \mathcal{A}$	top n	Section **??**
$T_D.f_D$:= **OP=Top** $T_S\|(\ldots).f_S\ \mathcal{A}$	top n	Section **??**
$T_D.\{f_V, f_X\}$:= **OP=ExistsIn** $T_S.f_V$	exists in	Section **??**

Table 5: Q Commands

Abbreviation	Explanation	$\mathcal{A}$
Sum	sum	—
Min	minimum	—
Max	maximum	—
NDV	number of distinct values	—
NumNull	number of null values	
ApproxNDV	approx number of distinct values	**TO BE COMPLETED**
ValAtIdx	value at specified index	Index
IdxAtVal	first index with specified value	Value

Table 6: Reduce operators

3.1 From Field to Scalar

Supports PTO

Note that there is an overlap between what is a field property (Table 4) and what is produced by reducing a field to a scalar.

Partial Table Specification now allowed for

1. NumNull
2. ValAtIdx. If no index with specified Value, returns -1
3. IdxAtVal. Index must be within bounds of table size.

3.2 Load Field from External Source

Valid values for **OP** are as follows

LoadCSV $\mathcal{A}$ to contain

1. DataFile. Name of Data File. MANDATORY
2. DataDirectory. Name of directory where data file is to be found. OPTIONAL. Default is current working directory.
3. FldType

LoadBin Same as above

3.3 Loading Table from External Sources

Valid values for **OP** are as follows.

LoadCSV In this case $\mathcal{A}$ contains

1. MetaDataFile. See Section **??**. MANDATORY
2. DataFile. Name of Data File. MANDATORY
3. DataDirectory. Name of directory where data file is to be found. OPTIONAL. Default is current working directory.
4. IgnoreHeader. Can have values true or false. Default is false. Ignores first line if set to true
5. FldSep. Can have values COMMA, TAB. Default is COMMA.
6. . . . **TO BE COMPLETED**

LoadBin In this case $\mathcal{A}$ contains

1. DataFile. Name of Data File. MANDATORY
2. DataDirectory. Name of directory where data file is to be found. OPTIONAL.
3. FldList. Comma separated list of field names e.g., `foo,bar`
4. FldSpec. Comma separated list of field types e.g., `I4,I8` Things to note
   (a) Number of fields must match number of format specifications
   (b) Formats SC and SV not allowed for LoadBin
   (c) Null values not allowed

LoadHDFS **TO BE COMPLETED**

3.4 From Scalars to Field

In all cases, $\mathcal{A}$ must contain **FldType**. Values provided must be consistent with field type. Valid values for **OP** are in Table 7. Details below.

Constant In this case, $\mathcal{A}$ must contain **Value**

Sequence In this case, $\mathcal{A}$ must contain **Start**, s, and **Increment**, δ. Values set to $f_0 = s, f_1 = s + \delta, f_i = (i - 1) \times \delta$

OP
Constant
Sequence
Period
Random

Table 7: stof: Valid values of **OP** for creating field from scalar

Period Like Sequence but $\mathcal{A}$ must also contain **Period**, T, which tells us how often to reset to initial value. For example, $s = 2, \delta = 3, T = 4$ means we get values like $\{2, 5, 8, 11, 2, 5, 8, 11, \ldots\}$

Random $\mathcal{A}$ must contain **Distribution**. If distribution is "Uniform", $\mathcal{A}$ must contain **Min** and **Max**.

3.5 Creating One Field from Another

Consider the creation of f_1 from f_2. See Table 8 for possible values of **OP**. If the field type of the newly created field, f_1, is the same as that of the source field, f_2, then a checkmark is placed in the column **InType = OutType**.

3.5.1 Shrink

Converts to smallest integer type that will not cause loss of precision. Valid input types are $\{I2, I4, I8\}$. Valid output types are $\{I1, I2, I4, I8\}$

3.5.2 Cast

Dangerous operation since it simply interprets the bytes differently. Valid transformations are in Table 9

3.5.3 Convert

- $\mathcal{A}$ must contain **NewFldType**
- If min/max of f_2 will cause overflow based on $FldType(f_1)$, then operation fails
- Loss of precision in conversions is for user to worry about

Abbreviation	$FldType(f_2)$	**Details**
Shrink	$\{I1, I2, I4, I8\}$	Section 3.5.1
Cast	$\{I1, I2, I4, I8, F4, F8\}$	Section 3.5.2
Convert	$\{I1, I2, I4, I8, F4, F8\}$	Section 3.5.3
BitCount	$\{I1, I2, I4, I8\}$	Section 3.5.4
Sqrt	$\{F4, F8\}$	square root
Abs	$\{I1, I2, I4, I8, F4, F8\}$	absolute value
CRC32	$\{I4, I8\}$	crc32 hash
!	$\{I1, I2, I4, I8\}$	Section 3.5.5
$++$	$\{I1, I2, I4, I8\}$	increment
$--$	$\{I1, I2, I4, I8\}$	decrement
˜	$\{I1, I2, I4, I8\}$	bit-wise complement
Reciprocal	$\{F4, F8\}$	
Accumulate	$\{I1, I2, I4, I8, F4, F8\}$	Section 3.5.6
Smear	I1	Section 3.5.7
Mix	I4, I8	Section 3.5.8
IdxWithReset	I1	Section 3.5.9

Table 8: f1opf2: Valid values of **OP** for creating one field from another

Old FldType	**New FldType**
I4	F4
F4	I4
I8	F8
F8	I8

Table 9: f1opf2: Valid Casts

3.5.4 Bit Count

$FldType(f_1) = I1$

3.5.5 Logical Not

! is the logical not operator.

3.5.6 Accumulate

- $f_1[0] = f_2[0]$
- $f_1[j] = \sum_{i=0}^{i=j} f_2[i]$

$FldType(f_1) \leftarrow FldType(f_2)$, except is over-ruled by user using NewFldType in $\mathcal{A}$

3.5.7 Smear

User needs to specify NumAfter, n_R, and NumBefore, n_L, in $\mathcal{A}$. This operation "smears" the selection, specified by source field f_2 by n_R to the right and n_L to the left i.e.,

- $f_1[i] = 1 \Rightarrow \forall j : 1 \leq j \leq n_R, f_2[i + j] \leftarrow 1$
- $f_1[i] = 1 \Rightarrow \forall j : 1 \leq j \leq n_L, f_2[i - j] \leftarrow 1$
- $n_R \geq 0$, $n_L \geq 0$, n_R and n_L cannot both be 0
- $FldType(f_1) \leftarrow FldType(f_2) = I1$
- $IsNulLFld(f_1) = false$

3.5.8 Mix

TO BE COMPLETED

3.5.9 Index with Reset

TO BE COMPLETED

3.6 Creating One Field from a Scalar and a Field

Here, we create a new field in a table by using an existing field and a scalar. See Table 10 for possible values of **OP**. $\mathcal{A}$must contain **SCALAR**.

3.6.1 Regex Match

$\mathcal{A}$ must contain **MatchType** which is either **Exact** or **Regex**. If the first, then its a straight equality comparison. Otherwise, it is considered a regex match. FldType of source field, f_2, is SC. FldType of newly created field is I1, values being 0 or 1.

3.7 Ternary Operator

There are three variants

1. $T_1(w := ?: x \;\; y \;\; z)$. In this case,
   - $FldType(x) = I1$
   - $HasNulLFld(x) = false$
   - y, z can be either a scalar or a field.
   - If both are fields, there field types must be the same. $FldType(w)$ is set to the field type of y or z
   - If one of them is a field, the scalar for the other should be compatible with the FldType of the field $FldType(w)$ is set to the FldType of the argument that is a field
   - If both are scalars, the FldType is chosen by the system to be smallest that will suffice.
   - No matter how chosen, $FldType(w) \in \{I1, I2, I4, I8, F4, F8\}$
2. $T_1(w := ?: lb \;\; ub \;\; y \;\; z)$
   - Constraints on y, z same as above
   - lb, ub are integers are specify a valid index range with $lb < ub$
   - Sets w in T to field y for all rows specified and to field z for all other rows.
3. $T_1(w := ?: lb \;\; ub \;\; val)$
   - lb, ub are integers are specify a valid index range with $lb < ub$
   - Field w must exist
   - val must be consistent with $FldType(w)$
   - Sets w in T to scalar s for all rows specified.
   - $FldType(w) \in \{I1, I2, I4, I8, F4, F8, SC\}$

3.8 f1f2opf3

Valid values for **OP** are in Table 11. Assume that we create $f_3 \leftarrow f_1 \oplus f_2$.

- $FldType(f_1) = FldType(f_2)$
- $FldType(f_3) \leftarrow FldType(f_1)$, except for concatenate. In that case,
  1. $FldType(f_1) = I1 \Rightarrow FldType(f_3) = I2$
  2. $FldType(f_1) = I2 \Rightarrow FldType(f_3) = I4$
  3. $FldType(f_1) = I4 \Rightarrow FldType(f_3) = I8$

4 xfer

- $Fldtype(f_I) \in I4, I8$
- $Fldtype(f_1) \leftarrow Fldtype(f_2)$
- $Fldtype(f_2) \in \{I1, I2, I4, I8, F4, F8, SC\}$
- $0 \leq min(f_I) \leq max(f_I) < |T_2|$

4.1 pack

1. $FldType(f_0) \leftarrow \{I4, I8\}$
2. Options must be specified as set of integer ranges such that
   (a) ranges are non-overlapping
   (b) minimum value of any range is 0
   (c) maximum value of any range is 31 for I4 and 62 for I8
   (d) All input fields must be non-negative
   (e) Each input field must be able to fit into the range allocated for it.

   Let option be $\{(l_1, u_1), (l_2, u_2), \ldots\}$. This means that

   (a) field f_1 will be shifted to occupy bits $[l_1, u_1]$ of f_0
   (b) field f_2 will be shifted to occupy bits $[l_2, u_2]$ of f_0
   (c) $\ldots$

4.2 fop

Following operations are supported

1. sort ascending
2. sort decending
3. permute (random)
4. saturate — set minimum or maximum value (or both) to user-specified

4.3 sortf1f2

Following operations are supported

1. sort first field ascending, second field don't care
2. sort first field ascending, second field ascending
3. sort first field descending, second field don't care
4. sort first field descending, second field descending

4.4 Append one table to another

TO BE COMPLETED

4.5 Percentiles

TO BE COMPLETED

4.6 Number in Range

TO BE COMPLETED

4.7 t1f1t2f2opt3f3

TO BE COMPLETED

4.8 Copy Table

TO BE COMPLETED

4.9 Copy Field

TO BE COMPLETED

4.10 Un-concatenate

TO BE COMPLETED

4.11 Create Field by "striding" another Field

- Options are
  1. **op** set to `stride`
  2. **start** set to integer in $[0, |T_2| - 1]$
  3. **incr** set to positive integer $< |T_2|/16$
- $FldType(f_2) \in \{I1, I2, I4, I8, F4, F8\}$
- $FldType(f_1) \leftarrow FldType(f_2)$
- $f_1[i] = f_2[(start + i \times incr) \bmod |T_2|]$

4.12 Create Field by "sub-sampling" another Field

- Options are
  1. **op** set to `subsample`
  2. **nR** set to positive integer, $0 < n_R < |T_2|/16$
  3. **replacement** set to `true` or `false`
- If T_1 exists, $n_R = |T_1|$
- $FldType(f_2) \in \{I1, I2, I4, I8, F4, F8\}$
- $FldType(f_1) \leftarrow FldType(f_2)$
- f_1 is created by picking items at random from f_2. Whether picks are made with or without replacement, depends on user specification

4.13 Counting

4.13.1 Counting (large NDV)

`count_vals`

4.13.2 Counting (small NDV)

`count_ht`

4.13.3 Counting (partial tables)

`countf`

4.14 Join

TO BE COMPLETED

4.15 dump

Arguments are

1. name of file into which tables are dumped
2. name of file into which fields are dumped

Creates CSV files of meta-data.
TODO: Deal with reference count incremeneting, checking, decrementing
TODO: Deal with external fields from scope creation
TODO: Setting foreign keys woith SetMeta
TODO: Scopes and all the mess that goes with it, including locking, ...

OP	I1	I2	I4	I8	F4	F8	SC	Details
+	✓	✓	✓	✓	✓			
−			✓	✓	✓			
*			✓	✓	✓			
/			✓	✓	✓			
%			✓	✓				
&	✓		✓	✓				
|	✓		✓	✓				
^	✓		✓	✓				
>	✓		✓	✓	✓			
<	✓		✓	✓	✓			
>=	✓		✓	✓	✓			
<=	✓		✓	✓	✓			
!=	✓		✓	✓	✓			
==	✓	✓	✓	✓	✓			
<<			✓	✓				
>>			✓	✓				
<=||>=			✓					
>&&<			✓					
>=&&<=			✓					
regex			✓				✓	Section 3.6.1

Table 10: Supported Operations for f1s1opf2

Abbreviation	Explanation	I1	I2	I4	I8	F4	F8
+		✓	✓	✓	✓	✓	✓
−		✓	✓	✓	✓	✓	✓
*		✓	✓	✓	✓	✓	✓
/		✓	✓	✓	✓	✓	✓
%		✓	✓	✓	✓		
&&		✓	✓	✓	✓		
||		✓	✓	✓	✓		
>		✓	✓	✓	✓	✓	✓
<		✓	✓	✓	✓	✓	✓
>=		✓	✓	✓	✓	✓	✓
<=		✓	✓	✓	✓	✓	✓
!=		✓	✓	✓	✓	✓	✓
==		✓	✓	✓	✓	✓	✓
..	concatenate	✓	✓	✓			
||!	or not	✓	✓	✓	✓		
&&!	and not	✓	✓	✓	✓		
&		✓	✓	✓	✓		
|		✓	✓	✓	✓		
^							
<<	left shift	✓	✓	✓	✓		
>>	right shift	✓	✓	✓	✓		

Table 11: f1f2opf3